1.
I guess the form of points in P is A [i] [j],
To avoid the left "maximal p" smaller than or equal to the right's because I want to use the biggest j for all A [i] and that will consume O(1) when I compare it with the previous maximal p, I start from the right side of the plane, which has the biggest I and smallest j for all maximal points.

We sort the array by the value of i through quicksort and that takes O(nlogn),
From the biggest i,
In each A [i], we select the biggest A [j], it takes O(n)
Then compare the value of j in new/left A [i], it should larger than the older/right j, this takes O(1) each time.
Until the end.

The above could find all maximal points but it already larger than O(nlogn). Hence, I don't know yet.

2.
You want to find a random number k, or the k is the i in the problem?
If it's the first one, I don't know, you need to check every number equal to k or not, and the time complexity is O(n). So, I don't know.

If it's the second one, you want to find the "peak" number.
Middle point m =n/2, left = [0⋯n/2], right = [n/2⋯n],
Select the middle number m = [n/2], check the A[m-1] and A[m+1],

If A[m-1] < A[m], and A[m+1] < A[m], the k = m;
If A[m] <A[m+1], then select the right side of m be the new array and repeat. Else, select the left side array.

Every time the array will be cut to nearly half, and we need one or two steps to compare the values around the m and m,
T (n) = T(n-1) + O(1), we will have logn levels at most. Hence, the time complexity is O(logn)

3.
Begin from the end of the array and maintain two variables below,
1.even length, which is the length of subsequence if length caused by the element comes at an even position.
2.odd length, which is the length of subsequence if length caused by the element comes at an odd position.
Set both to 1.

Beginning from the end of the array for an element a[i] scan all the elements a[j] where j>i

if a[j] <a[i] then

let odd length a[i] = max(oddLength(a[i]),evenLength(a[j])+1)

if a[j]>a[i] then

let even length a[i] = max(evenLength(a[i]),oddLength(a[j])+1)

return the maximum oddLength found so far.

It should be O(n^2), maybe